



Programación de Microservicios con Spring Boot y Red Hat SSO

Nivel Avanzado

Instructor: Carlos Carreño
ccarrenovi@gmail.com



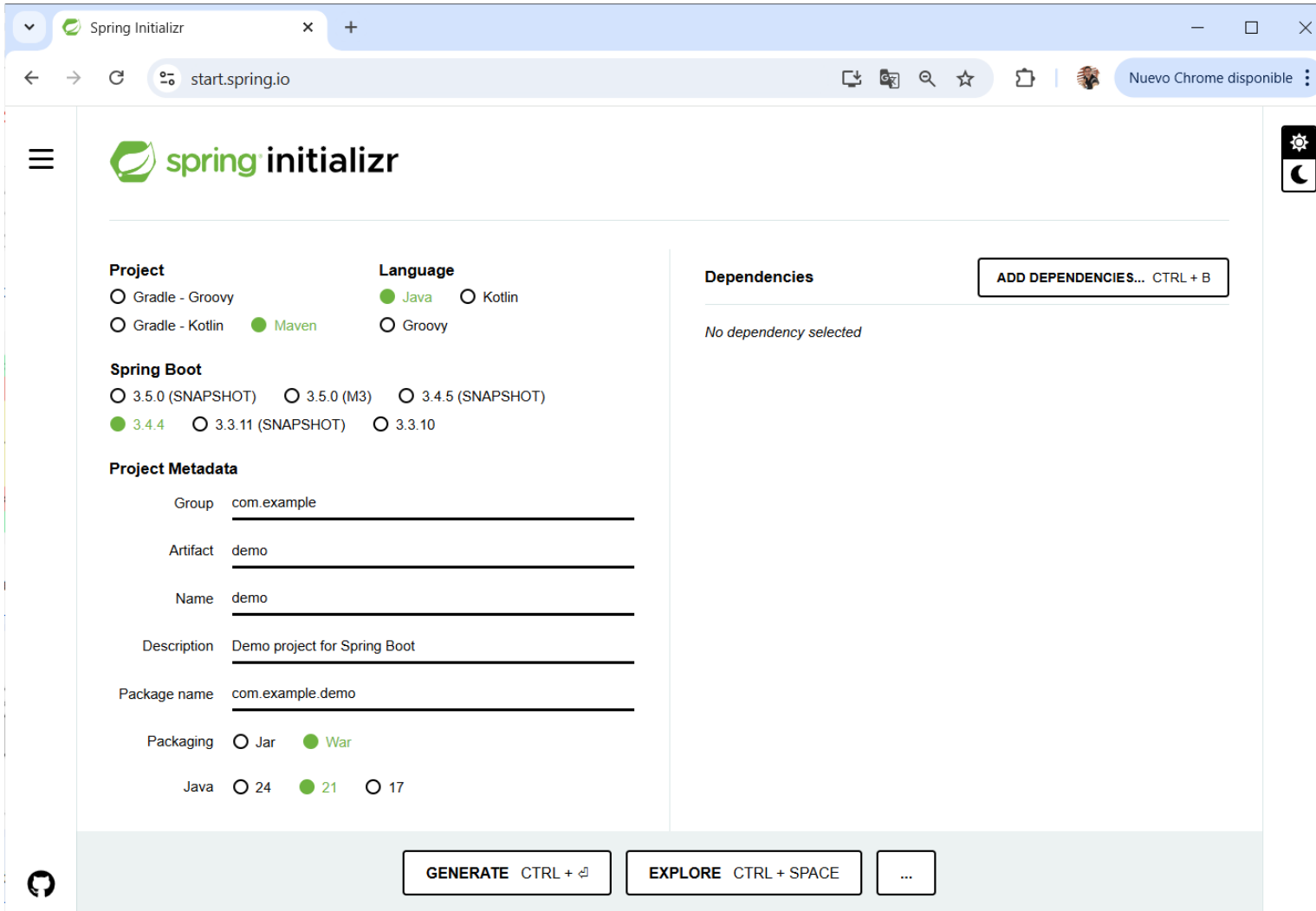
Modulo 2. Creación de Microservicios con Spring Boot

Objetivo: Aprender a desarrollar microservicios con Spring Boot 3, incluyendo configuración, comunicación entre servicios, seguridad y despliegue

Duración: 1h



Creación de Microservicios con Sprint Boot 3.X.X



The screenshot shows the Spring Initializr web application in a browser window. The browser's address bar shows the URL `start.spring.io`. The application interface includes a sidebar with a hamburger menu icon, the Spring logo, and a settings icon. The main content area is divided into several sections:

- Project:** Radio buttons for `Gradle - Groovy`, `Gradle - Kotlin`, and `Maven` (selected).
- Language:** Radio buttons for `Java` (selected), `Kotlin`, and `Groovy`.
- Spring Boot:** Radio buttons for versions `3.5.0 (SNAPSHOT)`, `3.5.0 (M3)`, `3.4.5 (SNAPSHOT)`, `3.4.4` (selected), `3.3.11 (SNAPSHOT)`, and `3.3.10`.
- Project Metadata:** Text input fields for `Group` (com.example), `Artifact` (demo), `Name` (demo), `Description` (Demo project for Spring Boot), and `Package name` (com.example.demo).
- Packaging:** Radio buttons for `Jar` and `War` (selected).
- Java:** Radio buttons for versions `24`, `21` (selected), and `17`.
- Dependencies:** A section with the text "No dependency selected" and a button labeled "ADD DEPENDENCIES... CTRL + B".

At the bottom of the interface, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and an ellipsis button "...".



Estructura de un proyecto Spring Boot

Estructura del proyecto

```
mi-servicio/  
|  
├─ src/main/java/com/ejemplo/miservicio/  
|   ├── MiServicioApplication.java      # Clase principal @SpringBootApplication  
|   ├── controller/  
|   |   └─ SaludoController.java        # REST Controller  
|   ├── service/  
|   |   └─ SaludoService.java           # Lógica de negocio  
|   └─ model/  
|       └─ Saludo.java                  # Modelo de datos  
|  
├─ src/main/resources/  
|   └─ application.properties           # Configuración  
|  
└─ pom.xml                             # Dependencias Maven
```



Controladores REST

- Un **Controlador REST** es una clase que gestiona solicitudes HTTP (GET, POST, PUT, DELETE, etc.) y devuelve datos (normalmente en formato JSON o XML) en lugar de una vista o página HTML.

```
@RestController
@RequestMapping("/api/saludo")
public class SaludoController {
    @GetMapping
    public String saludar() {
        return "Hola desde Spring Boot Microservicio!";
    }
}
```



Características del Controlador REST en Spring Boot

Característica	Descripción
 Anotación principal:	<code>@RestController</code> (combina <code>@Controller</code> + <code>@ResponseBody</code>)
 Define endpoints HTTP:	Utiliza anotaciones como <code>@GetMapping</code> , <code>@PostMapping</code> , <code>@PutMapping</code> , <code>@DeleteMapping</code> .
 Retorna datos:	Responde con JSON o XML (según configuración o cabecera HTTP <code>Accept</code>).
 Manejo de parámetros:	Extrae datos de la URL, cuerpo de la petición o parámetros usando <code>@PathVariable</code> , <code>@RequestParam</code> , <code>@RequestBody</code> .
 Puede incluir seguridad:	Integra con Spring Security, JWT, OAuth2, etc.



Uso de DTOs y manejo de respuestas JSON

- Un **DTO** es una clase simple que contiene atributos que representan la información que quieres enviar o recibir a través de la API.
- ✓ Evita exponer directamente las entidades JPA.
- ✓ Permite controlar qué datos viajan al cliente.
- ✓ Facilita validaciones (@Valid)



Ventajas de usar DTOs

Ventaja	Descripción
 Seguridad	Evita exponer entidades internas
 Precisión	Sólo envía datos necesarios
 Mantenibilidad	Facilita cambios sin afectar cliente
 Validación con Anotaciones	Ej: <code>@NotNull</code> , <code>@Size</code>



Validaciones y manejo de errores



DTO con anotaciones de validación

```
public class UsuarioRequestDTO {  
  
    @NotNull(message = "El nombre no puede ser nulo")  
    @Size(min = 2, max = 30, message = "El nombre debe tener entre 2 y 30 caracteres")  
    private String nombre;  
  
    @Email(message = "Debe ser un correo válido")  
    @NotBlank(message = "El correo es obligatorio")  
    private String correo;  
  
    @NotBlank(message = "La contraseña es obligatoria")  
    @Size(min = 6, message = "La contraseña debe tener al menos 6 caracteres")  
    private String contrasena;  
  
    // Getters y Setters  
}
```



continuación

Controlador usando @Valid

```
@RestController
@RequestMapping("/api/usuarios")
public class UsuarioController {

    @PostMapping
    public ResponseEntity<String> crearUsuario(@Valid @RequestBody UsuarioRequestDTO usuario) {
        // Si pasa validaciones, continúa aquí:
        return ResponseEntity.ok("Usuario creado con éxito: " + usuario.getNombre());
    }
}
```



continuación

- Manejo de errores de Validación

@RestControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<Map<String, String>> manejarErroresValidacion(MethodArgumentNotValidException ex) {

Map<String, String> errores = new HashMap<>();

ex.getBindingResult().getFieldErrors().forEach(error ->
 errores.put(error.getField(), error.getDefaultMessage())
);

return ResponseEntity.badRequest().body(errores);

}

}



Lab

- Ejercicio práctico: Primer microservicio RESTful

